bigger installation scenarios—by an installation tool using the deployment tool through this API.

[0016] The GUI can include a navigation tree window, a deployment parameters window, a legend window, an information window, and a repository preview. The navigation tree window can include a hierarchical display of catalogs and archives that a user may select for deployment. The deployment parameters window can include a display of parameter groups necessary for deployment of an archive. The legend window can include a display of a description of icons used in a deployment process. The information window can include a display of information for archives or catalogs or both that are selected from the navigation tree window. The repository window can include a preview of a deployment repository after deployment.

[0017] The invention can be implemented to realize one or more of the following advantages. The software deployment tool can deploy software updates on a system in conjunction with a system's current configuration so that dependencies are managed. The tool can match a current system configuration with an updated configuration, and identify conflicts between the current configuration and the update. The tool can handle software changes, including file update deployment, database schema deployment and other update package deployment. The tool can be implemented using factories and interfaces for the handling of software types and software server types, and so have an architecture that is extensible by new software types and software server types. Delivery archives contain software changes and additional information for a specific target system type. Archives can be implemented to extend the deployable archive types from the J2EE specification (.ear, .war, .rar) in a fully J2EE compliant way. The additional information that extends the archive types is embedded within such archives using manifest files and an additional deployment descriptor. Thus, such an archive satisfies the Java standard and extends it.

DESCRIPTION OF THE DRAWINGS

[0018] **FIG. 1** is a block diagram of a system.

[0019] **FIG. 2** is a block diagram of a software data archive data structure.

[0020] **FIG. 3** is a flow diagram of a delivery process.

[0021] **FIG. 4** is a block diagram of a supplementary manifest.

[0022] **FIG. 5** is a flow diagram of a process for selecting a deployment engine.

DETAILED DESCRIPTION

[0023] As shown in **FIG. 1, a** target computer system **10** includes a J2EE server **12** and can run Software Delivery Manager (SDM) **14**, an implementation of a deployment tool made in accordance with the invention. The computer system **10** can be a test system, a quality assurance system, a production system, or a combination of them, or any other kind of data processing system on which software is deployed.

[0024] SDM uses a repository **16**. SDM manages and deploys software delivery archives (SDAs)—the smallest units of software modification—that are received for deployment into the system **10**. SDAs might be received, for example, from a development system **18**, on any suitable carrier. The SDAs can be delivered in a software delivery catalog. A catalog contains meta-data descriptions of its SDAs, which the catalog bundles as logical deployment units.

[0025] Illustrated in **FIG. 2, a** software delivery archive (SDA) **50** is a data structure having a delivery format for software applications and software elements. The SDA **50** is a ZIP and JAR-compatible archive format that can be used as a container for other archives. JAR (Java Archive) is a platform-independent file format used for aggregating many files into one. For example, multiple Java applets along with their requisite components (e.g., class files, images, property files) can be bundled in a JAR file.

[0026] The SDA **50** contains a standard JAR manifest **52** and a supplementary manifest **54**, which contains additional context information, such as dependency information. The SDA also contains an SDA deployment descriptor **56**, with software type specific deployment configuration data, and one or more software files **58** for deployment.

[0027] An SDA that does not contain any other archives is an atomic SDA. This is also the smallest unit for deployment and patch delivery. However, SDAs can be nested, with the manifest information being passed on to the SDA at the highest level. SDA **50** is illustrated as including one nested software delivery archive, SDA'**60**. SDAs can be combined to a logical unit using a software delivery catalog, which is a special kind of SDA that contains only references to other SDAs.

[0028] A human deployer will generally be responsible for configuring and deploying software packages on the system **10** (**FIG. 1**). A deployer typically configures the software to be deployed, causes the configured software to be installed on the system in the appropriate places, and may start the software running. The additional deployment descriptor **56** of the SDA addresses the problem that deployment into J2EE can require a lot of interaction between the deployer and the J2EE engine. Rather than repeating this interaction during each deployment, the SDA **50** carries information from software development in the SDA deployment descriptor **56** that facilitates the deployment process. This is a transfer of application knowledge and developer expertise to the customer site, allowing a deployer without deep application knowledge to achieve successful and rapid deployment.

[0029] SDM **14** triggers the deployment of the SDAs at the customer site. Depending on the software type of the SDA to be deployed and on the target system's server type, SDM **14** uses a specific engine to carry out the deployment. These engines (or factories) are implemented for each software server type SDM **14** supports. As illustrated in **FIG. 5**, SDM receives (**70**) an SDA, determines (**72**) its software type, determines the target server type (**74**), and selects (**76**) a factory or engine to perform the deployment. Servers are generally typed in the repository **16** when they are added. This can be done either manually or through an API function of SDM **14**. The server type for deployment is identified through the element SoftwareType in the deployment descriptor.

[0030] For example, for deployment to a J2EE application server, the corresponding engine interacts with the J2EE